JOURNAL of COMPUTER SCIENCES INSTITUTE

JCSI 6 (2018) 97-101
Wysłane: 2017-11-15
Przyjęte: 2017-11-20

Wydajność pracy z bazami danych w aplikacjach ASP.NET MVC

Paweł Borys*, Beata Pańczyk
Politechnika Lubelska, Instytut Informatyki, Nadbystrzycka 36B, 20-618 Lublin, Polska

Streszczenie. Podczas tworzenia aplikacji internetowych działających w oparciu o bazę danych niezwykle ważne jest wybranie odpowiedniego narzędzia pozwalającego na obsługę bazy. Wybór ten ma wpływ zarówno na działanie wdrożonego już programu jak i na przebieg procesu jego wytwarzania. Jednak liczba dostępnych rozwiązań jest duża i często niełatwo jest zdecydować, z którego narzędzia najlepiej skorzystać w danym projekcie. Niniejszy artykuł przedstawia porównanie wydajności pracy z trzema popularnymi rozwiązaniami dla platformy ASP.NET MVC: ADO.NET, Entity Framework i NHibernate..

Słowa kluczowe: bazy danych; Entity Framework; NHibernate; ADO.NET; ASP.NET MVC

*Autor do korespondencji.

Adres e-mail: pawel13bor@gmail.com

ASP.NET MVC database applications performance

Paweł Borys*, Beata Pańczyk
Institute of Computer Science, Lublin University of Technology, Nadbystrzycka 36B, 20-618 Lublin, Poland

Abstract. During web application development, working with a database and choosing appropriate tool for database managing is extremely important. This choice influences both final application and its development process. However there is a large number of possibilities of choosing the best tool to use in a project. It often provides some difficulties. The main goal of this paper is to present the advantages and disadvantages of three popular solutions for ASP.NET MVC platform (ADO.NET, Entity Framework, NHibernate) to help make the proper decision.

Keywords: databases; Entity Framework; NHibernate; ADO.NET; ASP.NET MVC

*Corresponding author.

E-mail address: pawel13bor@gmail.com

1. Wstęp

Tworzenie nowoczesnych aplikacji internetowych nieodłącznie wiąże się z wykorzystaniem baz danych. W popularnym wzorcu architektonicznym Model-Widok-Kontroler (MVC), organizującym podział aplikacji na trzy części o odrębnych zadaniach, model odpowiada danym, na których pracuje program i które zazwyczaj przechowywane są w relacyjnej bazie danych [1]. Programista wykorzystując framework ASP.NET MVC stoi przed koniecznością wyboru efektywnej metody obsługi bazy danych. W przypadku skomplikowanej struktury relacyjnej bazy danych, w której tabele są połączone wieloma wzajemnymi relacjami, samodzielna implementacja tej funkcjonalności może okazać się bardzo czasochłonna, a w przypadku mniej doświadczonych programistów, również problematyczna. Dlatego tak ważnym zagadnieniem jest dobór odpowiedniego narzędzia wspomagającego programistę w tworzeniu aplikacji bazodanowych.

Celem niniejszego artykułu jest porównanie trzech popularnych narzędzi ułatwiających obsługę bazy danych w aplikacji ASP.NET MVC. Określone zostaną wady oraz zalety każdego z nich pod względem wydajności działania. Dodatkowo oceniony zostanie poziom skomplikowania kodu w języku C#, niezbędnego do implementacji danego rozwiązania. Przeprowadzone badanie pozwoli na łatwiejsze dobranie odpowiedniego narzędzia przez programistę przystępującego do tworzenia aplikacji internetowej w architekturze MVC. Ponieważ warstwa modelu aplikacji jest podstawą, na której opiera się implementacja jej logiki biznesowej, decyzje podjęte na tym etapie są niezwykle istotne dla całego projektu. Poza różnicami w samym procesie implementacji obsługi danych, mogą one spowodować duże różnice w wydajności działania finalnego programu.

Narzędzia porównywane w artykule to Entity Framework, NHibernate oraz ADO.NET. Dwie pierwsze z wymienionych technologii realizują nowocześniejsze, obiektowe podejście do bazy danych jakim jest mapowanie obiektowo-relacyjne (ang. Object-Relational Mapping, ORM). Ideą mapowania obiektowo-relacyjnego jest wykorzystanie korzyści jakie daje paradygmat programowania obiektowego do zarządzania danymi w relacyjnej bazie danych [2]. Z kolei ADO.NET jest starszym rozwiązaniem, w którym programista komunikuje się z serwerem bazodanowym bezpośrednio za pomocą zapytań SQL. Wszystkie z wymienionych technologii są aktualnie stosowane. Zarówno Entity Framework, jak i NHibernate są z powodzeniem wykorzystywane przez programistów i żadna z tych technologii nie jest uznawana za jednoznacznie lepszą [3].

2. Narzędzia pracy z bazami danych w ASP.NET MVC

2.1. ADO.NET

ADO.NET jest zbiorem bibliotek pozwalających na komunikację z bazą danych z poziomu klas języka C# [4]. Odbywa się to za pomocą pisanych przez programistę zapytań SQL, czego skutkiem jest szereg potencjalnych problemów, które mogą pojawić się w projekcie wykorzystującym tę technologię. Zmiany w strukturze tabel bazy danych mogą spowodować konieczność wprowadzania poprawek do kodu aplikacji w wielu miejscach, co jest zarówno czasochłonne jak i znacząco zwiększa ryzyko popełnienia błędu. Ponadto problematyczne może okazać się zaimplementowanie bardziej złożonych operacji na danych, obejmujących np. operacje kaskadowe.

2.2. Narzędzia ORM

Odpowiedzią na problemy technologii takich jak ADO.NET było zastosowanie mapowania obiektowo-relacyjnego. Narzędzia ORM prezentują dane przechowywane w tabelach bazy danych oraz relacje zachodzące między nimi w postaci klas obiektowego języka programowania. Takie podejście umożliwia operowanie na danych z wykorzystaniem szerokiego wachlarza możliwości programowania obiektowego. Jest to na przykład możliwość obsługi skomplikowanych, nierzadko wielokrotnie zagnieżdżonych powiązań pomiędzy obiektami. Ponadto frameworki takie jak **Entity Framework** lub **NHibernate** stanowią dodatkową warstwę abstrakcji oddzielającą kod aplikacji od bazy danych co ułatwia dostosowanie kodu programu do modyfikacji struktury bazy lub zmiany systemu zarządzania bazą danych.

3. Porównanie narzędzi

3.1. Środowisko testowe

Do testów wykorzystano autorską aplikację ASP.NET MVC stworzoną na potrzeby niniejszego badania. Wersje wykorzystanych narzędzi oraz specyfikacja środowiska testowego przedstawione są w tabelach 1 i 2.

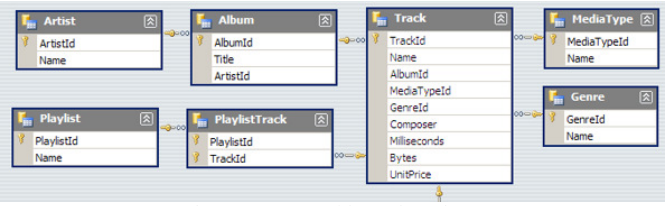
Tabela 1. Wykorzystane oprogramowanie.

Narzędzie	Wersja
.Net Framework	4.5.2
MSSQL	2016
Serwer IIS	7.5
MySQL	6.5.21
Serwer Apache	2.4.26
ADO.NET	4.6.1
Entity Framework	6.1.3
NHibernate	4.1.1

Tabela 2. Platforma testowa.

System	Microsoft Windows 7 64-bit
CPU	Intel Core i3
Pamięć RAM	4 GB

Bazą danych wykorzystaną do przeprowadzenia testów jest dostępna na licencji MIT baza Chinook [5], która reprezentuje sklep z utworami muzycznymi w wersji cyfrowej. Fragment diagramu ERD przedstawiono na rysunku 1. Wykorzystane w badaniach tabele zawierają od 300 do ok. 3500 rekordów.



Rys.1. Fragment schemantu testowej bazy danych.

3.2. Scenraiusze testów

W celu dokładnego porównania możliwości oraz ograniczeń badanych technologii przygotowano 6 scenariuszy testowych. Obejmują one podstawowe, często wykonywane operacje bazodanowe, tj. odczyt oraz zapis rekordów w bazie danych. Odczyt rekordów został zrealizowany na dwa sposoby. Pierwszym było wielokrotne wywołanie zapytania zwracającego rekordy o kolejnych wartościach pola *Id*, a drugim określenie warunku w skutek którego wynik pojedynczego zapytania składał się z grupy rekordów. Każda operacja została zrealizowana zarówno dla pojedynczej tabeli jak i dla grupy tabel powiązanych ze sobą wzajemnymi relacjami. Dzięki temu możliwe jest zbadanie w jakim stopniu w praktyce spełnione jest założenie, iż narzędzia ORM wspomagają pracę z rozbudowanymi strukturami danych w bazie. Ponadto każdy scenariusz został zrealizowany na bazie działającej na serwerze MSSQL (tworzony przez Microsoft) oraz MySQL (rozwijany przez Oracle). Pozwala to zaobserwować jak poszczególne narzędzia dostosowane są do współpracy z systemem baz danych innym niż ten tworzony przez firmę Microsoft. Wszystkie scenariusze testowe zostały przedstawione w tabeli 3.

Tabela 3. Scenriusze testów.

Oznaczenie	Scenariusz
S1	Odczyt rekordów z pojedynczej tabeli.
S2	Odczyt rekordów z 4 powiązanych tabel.
S3	Grupowy odczyt rekordów z pojedynczej tabeli.
S4	Grupowy odczyt rekordów z 4 powiązanych tabel.
S5	Wstawianie rekordów do pojedynczej tabeli.
S6	Wstawianie rekordów do 4 powiązanych tabel.

3.3. Aplikacja testowa

Do realizacji testów utworzono sześć projektów, po dwa dla każdego frameworka, odpowiednio dla bazy danych MSSQL oraz MySQL. Każdy z projektów zawierał klasę *HomeController.cs*, w której znajdowały się metody odpowiedzialne za wykonanie każdego ze scenariuszy. Dla tych dotyczących odczytu danych z bazy, liczba rekordów w pojedynczym pomiarze wynosiła jeden tysiąc. W przypadku zapytań warunkowych, były one dobrane tak aby zwrócić dokładnie tysiąc wierszy wynikowych. Dla scenariuszy zakładających wstawianie danych do bazy, liczba rekordów wynosiła sto. Prezentowane wartości wynikowe to średnie arytmetyczne wartości otrzymanych dla stu powtórzeń każdego ze scenariuszy.

Porównanie wydajności działania badanych narzędzi zostało zrealizowane poprzez zarejestrowanie czasu wykonania zadanych operacji oraz maksymalnej ilości pamięci operacyjnej zajętej przez aplikację podczas ich wykonywania. Do implementacji pomiarów zostały wykorzystane biblioteki udostępnianie przez środowisko .Net [6,7], co zostało przedstawione w przykładzie 1.

Przykład 1. Fragment kodu odpowiedzialny za pomiar czasu wykoania oraz wykorzystanej pamięci.

```
public void Start()
{   GC.Collect();
    GC.WaitForPendingFinalizers();
    GC.Collect();
    _startMemory = GC.GetTotalMemory(true);
    _watch.Reset();
    _watch.Start();
}
public void Stop()
{   _watch.Stop();
    milliseconds = _watch.Elapsed.TotalMilliseconds;
    memory = GC.GetTotalMemory(false) - _startMemory;
}
```

Implementacja scenariuszy została zrealizowana z wykorzystaniem domyślnej konfiguracji badanych narzędzi. Dla Entity Framework zastosowano podejście Database First, które polega na automatycznym wygenerowaniu przez framework odpowiednich klas języka C#, na podstawie wskazanej bazy danych. W przypadku NHibernate pliki konfiguracyjne w formacie XML zostały utworzone ręcznie. Wykorzystanie ADO.NET nie wymagało tworzenia dodatkowych plików konfiguracyjnych. Na przykładach 2-4 przedstawiono fragmenty kodu realizujące scenariusz S2.

Przykład 2. Realizacja scenariusza S2 dla ADO.NET.

```
for (int j = 1; j <= 1000; j++)
{   command.CommandText = "SELECT t.TrackId, t.Name,
t.Composer, t.Milliseconds, t.Bytes, t.UnitPrice, al.AlbumId AS
al_Id, al.Title AS al_Title, g.GenreId AS g_Id, g.Name AS
g_Name, mt.MediaTypeId AS mt_Id, mt.Name AS mt_Name "
+   "FROM Track t " +
    "JOIN Album al ON al.AlbumId = t.AlbumId " +
    "JOIN Genre g ON g.GenreId = t.GenreId " +
    "JOIN MediaType mt ON mt.MediaTypeId = t.MediaTypeId
" +  "WHERE t.TrackId = " + j.ToString();

    SqlDataReader reader = command.ExecuteReader();
    while (reader.Read())
    {
     Track track = new Track()
     {
         Id = (int)reader["TrackId"],
         Name = reader["Name"].ToString(),
         Bytes = (int)reader["Bytes"],
         Composer = reader["Composer"].ToString(),
         Milliseconds = (int)reader["Milliseconds"],
         UnitPrice = (decimal)reader["UnitPrice"]
     };
     tracks.Add(track);
     Album album = new Album()
     {
         Id = (int)reader["al_Id"],
         Title = reader["al_Title"].ToString()
     };
     albums.Add(album);
    Genre genre = new Genre()
     {
         Id = (int)reader["g_Id"],
         Name = reader["g_Name"].ToString()
     };
     genres.Add(genre);
     MediaType mediaType = new MediaType()
     {
         Id = (int)reader["mt_Id"],
         Name = reader["mt_Name"].ToString()
     };
     mediaTypes.Add(mediaType);
}
reader.Close();
}
```

Przykład 3. Realizacja scenariusza S2 dla Entity Framework.

```
for (int j = 0; j < 1000; j++)
{
    tracks.Add(context.Tracks.Where(t => t.TrackId == j +
1).Single());
    albums.Add(tracks[j].Album);
    genres.Add(tracks[j].Genre);
    mediaTypes.Add(tracks[j].MediaType);
}
```

Przykład 4. Realizacja scenariusza S2 dla NHibernate.

```
for (int i = 0; i < 1000; i++)
{
    tracks.Add(session.Get<Track>(i + 1));
    albums.Add(tracks[i].Album);
    genres.Add(tracks[i].Genre);
    mediaTypes.Add(tracks[i].MediaType);
}
```

Zaprezentowane fragmenty kodu wyraźnie pokazują jak bardzo narzędzia ORM upraszczają pracę z bazą danych. Używając ADO.NET konieczna jest ręczna obsługa połączenia z bazą przez programistę oraz skonstruowanie i wywołanie zapytania SQL. W przypadku pracy z tabelami powiązanymi wzajemnymi relacjami, niezbędne jest również zadbanie o zachowanie tych relacji ładowania danych do obiektów w pamięci. Narzędzia wykorzystujące mapowanie obiektowo-relacyjne zwalniają programistę z odpowiedzialności za połączenie z bazą danych, utrzymanie relacji oraz konstruowanie zapytań. W efekcie, w powyższych przykładach oba narzędzia ORM potrzebowały wielokrotnie mniej linii kodu niż ADO.NET na realizację tej samej operacji odczytu. Pozwala to ograniczyć czas niezbędny na oprogramowanie pracy z bazą i skupić się na logice biznesowej aplikacji, podczas gdy to ORM odpowiada za cały proces, w wyniku którego dane z tabel bazy danych są mapowane do obiektowej struktury klas języka C#.

3.4. Wydajność

W tabelach 4-9 oraz na rysunkach 2-5 przedstawiono wyniki pomiarów z przeprowadzonych badań.

Tabela 4. Wyniki pomiarów dla scenariusza S1.

Baza danych	Narzędzie	Czas [s]	Pamięć [MB]
MSSQL	ADO.NET	107	1,29
	Entity Framework	918	3,99
	NHibernate	576	4,01
MySQL	ADO.NET	199	3,07
	Entity Framework	1 183	3,98
	NHibernate	712	3,28

Tabela 5. Wyniki pomiarów dla scenariusza S2.

Baza danych	Narzędzie	Czas [s]	Pamięć [MB]
MSSQL	ADO.NET	144	5,11
	Entity Framework	1 242	5,36
	NHibernate	764	2,81
MySQL	ADO.NET	480	2,85
	Entity Framework	1 517	5,26
	NHibernate	885	4,14

Tabela 6. Wyniki pomiarów dla scenariusza S3.

Baza danych	Narzędzie	Czas [s]	Pamięć [MB]
MSSQL	ADO.NET	2	0,11
	Entity Framework	3	0,11
	NHibernate	40	1,40
MySQL	ADO.NET	3	0,15
	Entity Framework	4	0,21
	NHibernate	23	1,42

Tabela 7. Wyniki pomiarów dla scenariusza S4.

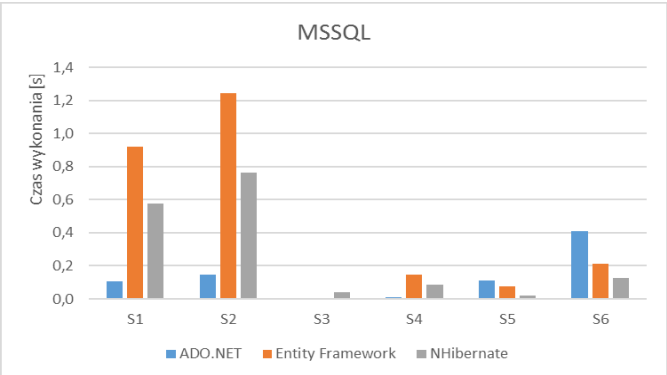
Baza danych	Narzędzie	Czas [s]	Pamięć [MB]
MSSQL	ADO.NET	12	0,52
	Entity Framework	147	5,27
	NHibernate	87	4,83
MySQL	ADO.NET	18	0,93
	Entity Framework	157	5,58
	NHibernate	78	4,08

Tabela 8. Wyniki pomiarów dla scenariusza S5.

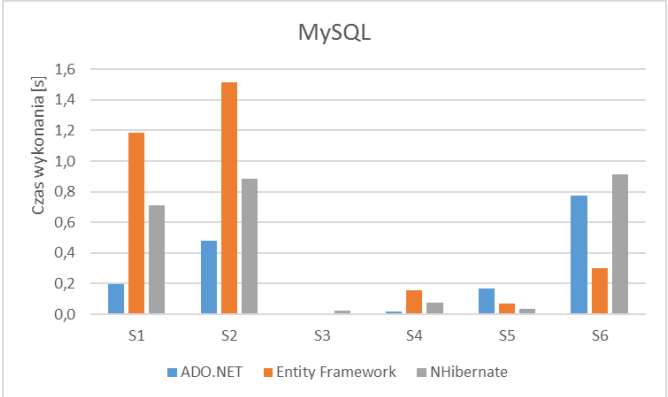
Baza danych	Narzędzie	Czas [s]	Pamięć [MB]
MSSQL	ADO.NET	112	0,08
	Entity Framework	77	1,88
	NHibernate	22	0,54
MySQL	ADO.NET	168	0,21
	Entity Framework	67	2,21
	NHibernate	34	0,48

Tabela 9. Wyniki pomiarów dla scenariusza S6.

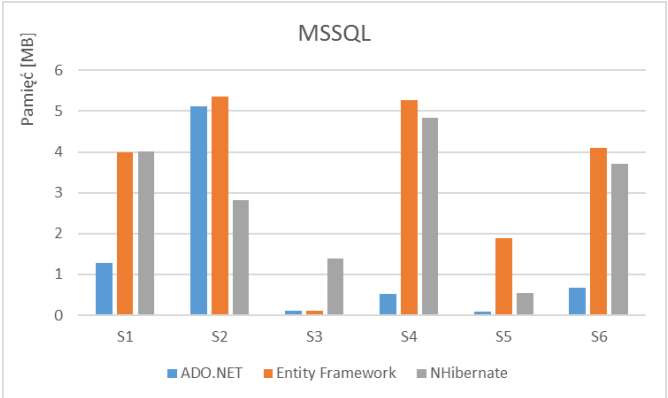
Baza danych	Narzędzie	Czas [s]	Pamięć [MB]
MSSQL	ADO.NET	410	0,68
	Entity Framework	213	4,10
	NHibernate	124	3,71
MySQL	ADO.NET	778	1,17
	Entity Framework	302	8,42
	NHibernate	914	12,46



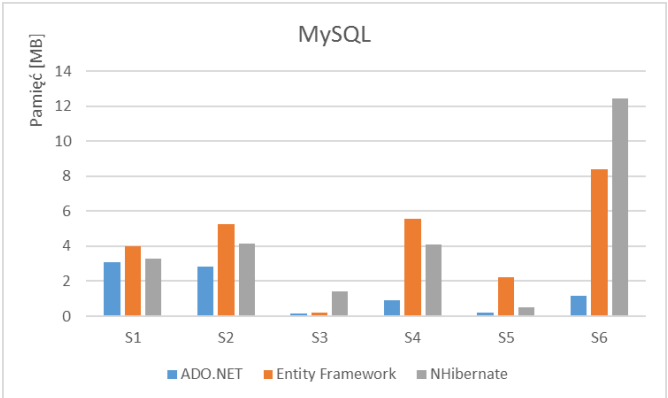
Rys. 2. Zestawienie wyników pomiaru czasu wykonania dla bazy MSSQL.



Rys. 3. Zestawienie wyników pomiaru czasu wykonania dla bazy MySQL.



Rys. 4. Zestawienie wyników pomiaru zajętej pamięci dla bazy MSSQL.



Rys. 5. Zestawienie wyników pomiaru zajętej pamięci dla bazy MySQL.

4. Wnioski

W artykule porównano trzy narzędzia służące do pracy z bazami danych z poziomu kodu aplikacji ASP.NET MVC. Z przeprowadzonej analizy wynika, że żadnego z badanych sposobów obsługi bazy danych – mapowania obiektowo-relacyjnego oraz podejścia opartego na ręcznym tworzeniu zapytań SQL – nie można określić jako lepszego od pozostałych bez wzięcia pod uwagę dodatkowych czynników. Są to zagadnienia takie jak stopień skomplikowania struktury bazy danych lub poziom wiedzy i doświadczenia programisty.

Spośród narzędzi ORM lepsze wyniki osiągał NHibernate, który w większości przypadków górował nad Entity Framework zarówno pod względem czasu jak i wykorzystanej

pamięci. Jedyne przypadki, w których NHibernate osiągnął gorsze wyniki to scenariusz S3 oraz wersja scenariusza S6 dla bazy MySQL. Był to również jedyny przypadek, w którym rodzaj bazy danych miał znaczący wpływ na wyniki badań. Najważniejszymi zaletami Entity Framework nie są wyniki tworzonych z jego pomocą aplikacji lecz stopień integracji tego narzędzia z .NET framework oraz środowiskiem Visual Studio. Bez konieczności instalacji dodatkowych narzędzi programista ma dostęp do graficznego edytora relacji oraz funkcji automatycznie generujących pliki konfiguracyjne. Ponieważ jednak istnieją narzędzia wspomagające NHibernate, które dają dostęp do takich samych funkcjonalności, według opinii autorów niniejszego artykułu NHibenate jest narzędziem lepszym w tym zestawieniu. Z kolei interfejs ADO.NET, pomimo wad opisanych podczas przedstawiania kodu aplikacji, zapewnił dużo szybsze wykonywanie zadań oraz zdecydowanie mniejsze zużycie pamięci we wszystkich przypadkach, poza tymi dotyczącymi wstawiania do bazy obiektów połączonych wieloma wzajemnymi relacjami.

Programista dokonując wyboru odpowiedniego narzędzia (np. pomiędzy ADO.NET a NHibernate) musi wziąć pod uwagę wymagania stawiane w stosunku do tworzonej aplikacji. Jeżeli najistotniejszym czynnikiem jest szybkość działania lub wykorzystana pamięć - odpowiednim wyborem jest ADO.NET. Natomiast w przypadku kiedy niezbędna jest większa elastyczność względem struktury bazy lub szybkość i łatwość implementacji - należy wybrać narzędzie ORM takie jak NHibernate.

Literatura

[1] Freeman A.: Pro ASP.NET MVC 5, Apress, 2013.

[2] Understanding Object-Relational Mapping: A Framework Based Approach. The International Journal on Advances in Software, 2009, nr 2 i 3.

[3] https://www.linkedin.com/pulse/orm-read-performance-ef-vs-dapper-balazs-hideghety, [10-11-2017]

[4] Bipin J. i in. Professional ADO.NET Programming, Apress, 2001

[5] https://chinookdatabase.codeplex.com, [10-11-2017]

[6] https://docs.microsoft.com/en-us/dotnet/api/system.diagnostics.stopwatch, [10-11-2017]

[7] https://docs.microsoft.com/en-us/dotnet/standard/garbage-collection/ [10-11-2017]